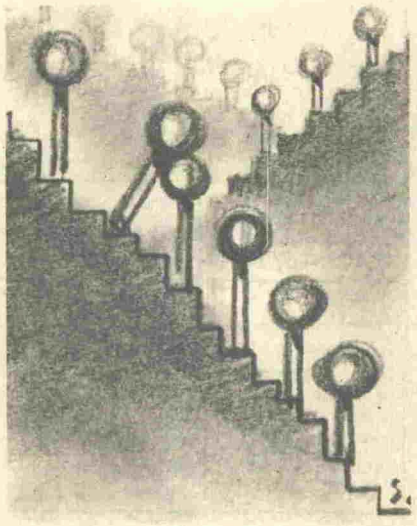
ภาษา C

จเร เลิศสุดวิชัย

การประยุกต์ใช้งาน แพคเกจลิงก์ลิสต์

กลเม็ดการจัดเรียงคิว โดยประยุกต์จาก แพคเกจลิงก์ลิสต์



มุมภาษาซีในฉบับนี้จะเป็นการนำเอาแพคเกจลิงก์ลิสต์มาประยุกต์ใช้งาน ซึ่งลักษณะโครงสร้างข้อมูลแบบลิงก์ลิสต์สามารถจะนำไปประยุกต์ใช้งานเพื่อสร้างโครงสร้างข้อมูลแบบต่าง ๆ ได้ เช่น คิว (Queue) สแตก (Stack) หรือแคช (Cache) เป็นต้น โดยจะเสนอการนำแพคเกจลิงก์ลิสต์มาสร้างคิวก่อน

คุณสมบัติของคิว

คิวเป็นโครงสร้างข้อมูลสำหรับจัดเก็บข้อมูลที่มีลักษณะของลำดับการดึงข้อมูลเป็นลำดับเดียวกับการนำข้อมูลเข้า ซึ่งส่วนใหญ่มักจะกล่าวว่า "First In First Out" (เข้าก่อนออกก่อน) หรือเขียนย่อเป็น FIFO

สำหรับคิวที่เราจะพัฒนาขึ้นด้วยแพคเกจลิงก์ลิสต์นี้จะเป็นแพคเกจคิวที่สามารถนำไปใช้งานได้อีก (Reusable) เช่นเดียวกับตัวแพคเกจลิงก์ลิสต์ที่ถูกนำมาใช้สร้างด้วย จากการพิจารณาจะพบว่ามีการกระทำกับโครงสร้างข้อมูลแบบคิวอยู่สองการกระทำ ซึ่งเป็นการกระทำพื้นฐานจริง ๆ ได้แก่ การดึงข้อมูลออก (ขอตั้งชื่อเป็น Qremove ()) และการนำข้อมูลเข้าคิว (Qinsert ()) โดยจากลักษณะของลิงก์ลิสต์ที่นำมาสร้างคิวเราจะพบว่า การนำข้อมูลเข้าคิวนั้นทำได้ง่าย ๆ คือ นำเอาข้อมูลมาต่อที่ส่วนหัวของลิสต์ ส่วนการนำข้อมูลออกก็ทำโดยการดึงข้อมูลออกจากท้ายของลิสต์นั่นเอง และสำหรับการกระทำพื้นฐานอย่างอื่นสามารถเรียกใช้งานจากแพคเกจลิงก์ลิสต์ได้โดยตรง ซึ่งอาจจะเปลี่ยนแปลงชื่อฟังก์ชันเหล่านั้นให้มาเป็นของคิว ดังในรูปที่ 1

ในตัวโมดูลของแพคเกจคิว เพื่อที่จะใช้บริการจากฟังก์ชันของแพคเกจลิงก์ลิสต์ จึงต้องประกาศตัวแปรสแตติกที่เป็นโคบอลขึ้นมาสำหรับเป็นพอยน์เตอร์ของคิวในการทำงานในโมดูล โดยใช้ชื่อว่า q (เหมือนตัวแปร list ในแพคเกจลิงก์ลิสต์) ดังในรูปที่ 2

ฟังก์ชันแรกของคิวคือฟังก์ชัน Qinsert การทำงานภายในฟังก์ชันนี้เป็นการนำข้อมูลเข้าไปเก็บในส่วนหัวของลิสต์เท่านั้นเอง โดยเพิ่มการตรวจสอบเงื่อนไขว่าถ้าคิวว่างจะต้องเรียกฟังก์ชัน LLinit ก่อน ซึ่งฟังก์ชัน Qin-

sert แสดงไว้ในรูปที่ 3 ส่วนฟังก์ชัน Qremove แสดงในรูปที่ 4 การทำงานภายในฟังก์ชันจะทำหน้าที่ดึงข้อมูลออกจากลิสต์ทางส่วนท้ายของลิสต์

ประยุกต์ใช้งานแพคเกจคิว

หลังจากที่เราได้พัฒนาแพคเกจคิวโดยอาศัยแพคเกจของลิงก์ลิสต์สำเร็จแล้ว ต่อไปจะเป็นตัวอย่างการนำแพคเกจคิวไปใช้งาน ปัญหาต่าง ๆ ที่จะต้องแก้ไขด้วยคิวนั้นต้องเป็นปัญหาที่มีคุณสมบัติตรงกับคิวคือ FIFO ยกตัวอย่างเช่น

การค้นหาข้อความ

ปัญหาแรกที่จะขอกล่าวเป็นตัวอย่างคือ เรื่องการค้นหาข้อความบางคำจากกลุ่มข้อความจำนวนหนึ่ง เช่น ต้องการค้นหาคำว่า "FIFO" ในกลุ่มข้อความกลุ่มหนึ่ง ดังรูปที่ 5 เมื่อพบคำว่า "FIFO" ในกลุ่มข้อความของบรรทัดใดจะต้องพิมพ์ข้อความบรรทัดนั้นออกมา พร้อมกับพิมพ์ข้อความของบรรทัดก่อนหน้าอีก 2 บรรทัด โดยการพิมพ์ข้อความออกมาแต่ละบรรทัดจะต้องเรียงลำดับบรรทัดตามเดิมด้วย ดังรูปที่ 5

จากลักษณะของปัญหาพบว่าสามารถนำเอาคิวมาช่วยแก้ปัญหาได้ ซึ่งจะจำลองการแก้ไขปัญหานี้ด้วยคิวดังในรูปที่ 6 โดยจะจำกัดคิวให้สามารถ

```
/* header file of Queue package */

#include "lldef.h"


#define Qset            LLsetlist /* Set current queue */
#define Qitemsize         LLsetsize /* define size of item in queue */
#define Qlength         LLlength    /* Return number of item in queue */

#define QUEUE           LINKLIST


/*----- Prototype of Queue pakage ----*/

void Qinsert (char *newitem);
void Qremove (char *newitem);
```

รูปที่ 1 ไฟล์ queuedef.h

```
/*---- File : queue.c ----*/
/* Module of queue package */
#include <stdio.h>
#include "queuedef.h"

/* Globol variable uses in queue modlue */
static struct QUEUE *q;
```

รูปที่ 2 ภายในโมดูลของคิวต้องประกาศตัวแปรสแตติก q เป็นพอยน์เตอร์ เพื่อใช้งานในโมดูล

```
void Qinsert (newitem)
char *newitem;
{
  if (LLlength () == 0)
    LLinit (newitem);
  else
    LLaddhead (newitem);
}
```

รูปที่ 3 ฟังก์ชัน Qinsert ซึ่งจะทำหน้าที่นำข้อมูลเข้าไปเก็บในคิว (โดยเก็บไว้ที่ส่วนหัวของลิสต์)

```
void Qremove (olditem)
char *olditem;
{
  LLtail ();
  LLretrieve (olditem);
  LLdelete ();
}
```

รูปที่ 4 ฟังก์ชัน Qremove ทำหน้าที่นำข้อมูลออกจากคิว (โดยนำข้อมูลออกทางส่วนท้ายของลิสต์)

```
A queue is a storage mechaniism in which items are retrieved in the as they
were entered. The next item retrieved is the one that resided in the storage
the longest. This kind of retrieval is often referred to as FIFO, for First
In First Out. Queues conjure up pictures of people on line for tickets at a
theater. The next person off the line is the on who was on line the longest.

*** Found the substring ***
A queue is a storage mechaniism in which items are retrieved in the as they
were entered. The next item retrieved is the one that resided in the storage
the longest. This kind of retrieval is often referred to as FIFO, for First
```

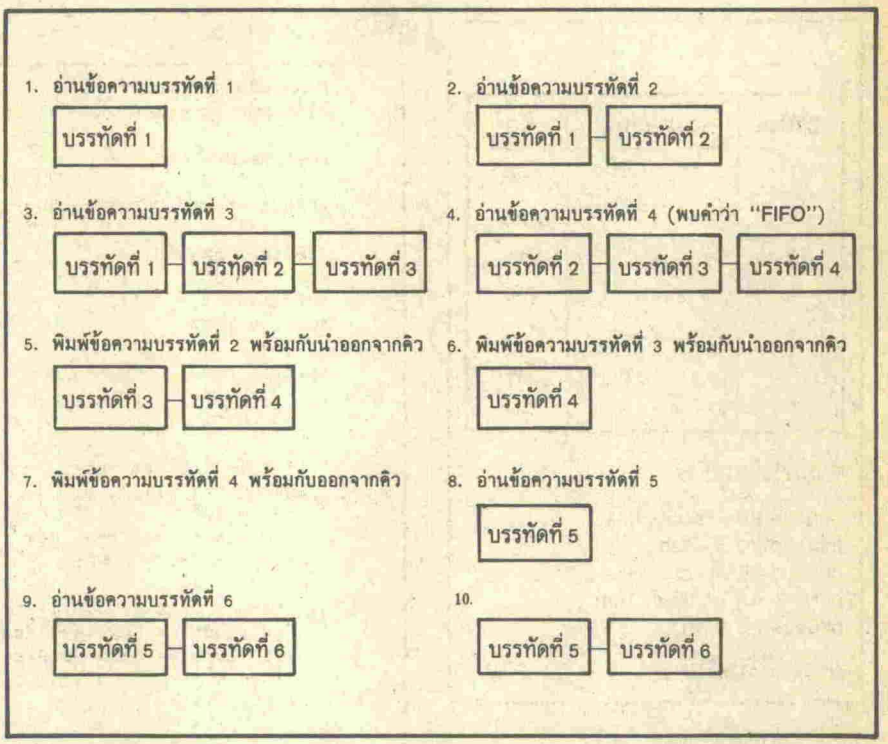
รูปที่ 5 แสดงตัวอย่างข้อความทั้งหมดที่ต้องการค้นหาคำว่า "FIFO" ซึ่งจะพบว่าอยู่ในบรรทัดที่ 4 ของข้อความ

ก็บข้อความได้เพียง 3 บรรทัด ทุกครั้ง
อ่านข้อความเข้ามาแต่ละบรรทัดจะ
ก็บไว้ในคิว ถ้าคิวเต็ม 3 บรรทัดจะดึง
รรทัดที่เก็บในคิวตอนต้นออก แล้ว
งเก็บข้อความบรรทัดใหม่ต่อท้ายลงไป
ละเมื่อพบคำว่า "FIFO" ในข้อความ
รรทัดใดก็จะพิมพ์ข้อความที่เก็บไว้
นคิวออกมาจนหมดคิว จากรูปที่ 6
ามารถเขียนโปรแกรมได้ดังรูปที่ 7

ัดคิวคนไข้ในโรงพยาบาล

คนไข้ที่ใช้บริการจากโรงพยาบาล
หรือคลินิกจะต้องเรียงลำดับเข้ารับการ
กษา ซึ่งคนไข้หรือผู้ป่วยแต่ละราย
ะมีอาการหรือความเจ็บป่วยต่างกัน
ช่น บางคนเป็นไข้หวัดแต่มาถึงโรง-
ยาบาลก่อน ในขณะที่มีคนไข้อีกราย
ระสบอุบัติเหตุหัวแตกเป็นบาดแผล
หญ่มาก ในกรณีนี้คนไข้ที่หัวแตกจะ
ด้รับการรักษาก่อน ดังนั้นจะเห็นได้
าปัญหาเรื่องคนไข้ในโรงพยาบาลย่อม
ม่ใช่คิวแบบธรรมดา แต่เป็นคิวที่ข้อมูล
ต่ละตัวภายในคิวมีลำดับความสำคัญ
ม่เท่ากัน (Priority Queue)

ลองสมมติว่าถ้าจะทำการเขียน
ปรแกรมเพื่อจัดการให้คนไข้เข้ารับการ
กษาพยาบาล โดยนำแพคเกจคิวที่
ัฒนาแล้วมาใช้งาน เพื่อสร้างความ
ำใจปัญหา สมมติว่าคนไข้แต่ละราย
ะมีลำดับความสำคัญในการเข้ารับการ
กษาต่างกัน ตั้งแต่ 0 (ลำดับความ
าคัญสูงสำหรับคนไข้ที่มีอาการอยู่ใน
นอันตราย) จนถึง 4 (ลำดับความ
าคัญต่ำสุดสำหรับคนไข้ธรรมดา) การ
างานของโปรแกรมเริ่มจากการรับชื่อ
องผู้ป่วย ตลอดจนลำดับความสำคัญ
ามที่แพทย์กำหนด และเมื่อแพทย์
รักษาพร้อมจะทำการรักษาคนไข้ โปร-
กรมจะแสดงรายชื่อคนไข้รายต่อไป



รูปที่ 6 จำลองการแก้ไขปัญหาด้วยคิว

```
/*---- File Textm.c ----*/
#include <stdio.h>
#include <string.h>
#include "queuedef.h"

#define LINES2PRINT   3

main ()
{
  static struct QUEUE lineq;
  char line [80],oldline [80];
  int n;

  /* initialize queue. */
  Qset (&lineq);
  Qitemsize (sizeof (line));
  while (gets(line) != .0)
    {

    /* Add item to storage */
    Qinsert (line);
    if (Qlength () > LINES2PRINT)
      Qremove (oldline);

    /* If substring is present, drain queue. */
    if (strstr (line, "FIFO") != 0)
      {
      printf ("*** Found the substring ***\n");
      while (Qlength () > 0)
        {
        Qremove (oldline);
        printf ("%s\n",oldline);
        }
      }
    }
}
```

รูปที่ 7
โปรแกรมการค้นหาข้อความ

ลำดับ	ความสำคัญ	ชื่อ
1	2	John
2	3	Anne
3	0	Mary
4	3	Tom
5	0	Les
6	2	Sue
7	1	Stan

ตัวอย่างข้อมูล

pqueue [0] = Mary, Les
pqueue [1] = Stan
pqueue [2] = John, Sue
pqueue [3] = Anne, Tom
pqueue [4] = NULL

การแก้ข้อมูลในคิว

รูปที่ 8 ตัวข้อมูลคนไข้และการจัดเก็บ

ที่ควรได้รับการรักษา โดยเรียงลำดับตามความสำคัญและลำดับในคิว

การแก้ไขปัญหาเรื่องคนไข้ในโรงพยาบาล ซึ่งเป็นคิวที่มีลักษณะเป็น Priority Queue นั้น ถ้านำแพคเกจคิวมาช่วยจะต้องมองเป็นอะเรย์ของคิวโดยเรียงลำดับตามความสำคัญของคนไข้ ซึ่งใช้ชื่อว่า pqueue [ ] แล้วเพื่อความเข้าใจการเก็บข้อมูลคนไข้ผู้เขียนได้สมมติรายชื่อคนไข้พร้อมลำดับความสำคัญ และการจัดเก็บรายชื่อคนไข้ในคิวดังรูปที่ 8 ส่วนโปรแกรมนั้นจะอยู่ในรูปที่ 9 ซึ่งการทำงานของโปรแกรมจะแสดงเมนูให้เลือกการทำงานสามหัวข้อคือ การเพิ่มรายชื่อคนไข้ การเรียกคนไข้เพื่อเข้ารับการรักษา และเลิกโปรแกรมตัวอย่างการป้อนข้อมูลเข้าโปรแกรมจะแสดงในรูปที่ 10 ตอนท้ายผู้เขียนได้รวบรวมแพคเกจคิวเอาไว้ในรูปที่ 11 และรูปที่ 12

รูปที่ 9

```
#include <stdio.h>
#include "queuedef.h"

#define NQUEUES   5

static struct QUEUE pqueue [NQUEUES];

#define NAMELEN 40
#define ADD     '1'
#define GET     '2'
#define DONE    '3'

main()
{
  int todo;

  /*-- initialize --*/
  PqueueInit ();
  todo = 100;
  /*-- start looping.. ask, do, ask, do, etc. --*/
  while (todo != DONE)
    {
    todo = what_to_do ();
    if (todo == ADD) AddPatient ();
    if (todo == GET) GetPatient ();
    }
  exit (0);
}

PqueueInit ()
{
  int n;

  for (n=0; n<NQUEUES; n++)
    {
    Qset (&pqueue [n]);
    Qitemsize (NAMELEN);
    }
}

what_to_do ()
{
  printf ("1... To Add a Patient to Queue\n");
  printf ("2... To Get an Available  Patient\n");
  printf ("3... To Quit Program\n");
  printf ("1-3 :");
  return (getche ());
}

AddPatient ()
{
  char patient [NAMELEN];
  int n;

  printf ("\nWhat Priority ? (0-4) :");
  scanf ("%d",&n);
  Qset (&pqueue [n]);
  printf ("Patient Name to Add : ");
  scanf ("%s",patient);
  Qinsert (patient);
  printf ("\n");
}

GetPatient ()
{
  char patient [NAMELEN];
for (n=0; n<NQUEUES; n++)
  {
  Qset (&pqueue [n]);
  if (Qlength () >0)
    {
```

รูปที่ 9 (ต่อ)

```
        Qremove (patient);
        printf ("\nPriority: %d Next Patient: %s\n",n,patient);
        printf ("\n");
        return;
        }
      }
   printf ("\nNo patients waiting\n");
   printf ("\n");
}
```

รูปที่ 10

```
2... To Get an Available  Patient
3... To Quit Program
1-3 : 1
What Priority ? (0-4) : 2
Patient Name to Add : Andrew

1... To Add a Patient to Queue
2... To Get an Available  Patient
3... To Quit Program
1-3 : 1
What Priority ? (0-4) : 2
Patient Name to Add : John

1... To Add a Patient to Queue
2... To Get an Available  Patient
3... To Quit Program
1-3 : 1
What Priority ? (0-4) : 1
Patient Name to Add : Alice

1... To Add a Patient to Queue
2... To Get an Available  Patient
3... To Quit Program
1-3 : 2
Priorityty: 1 Next Patient: Alice

1... To Add a Patient to Queue
2... To Get an Available  Patient
3... To Quit Program
1-3 : 2
Priority: 2 Next Patient: Andrew

1... To Add a Patient to Queue
2... To Get an Available  Patient
3... To Quit Program
1-3 : 2
Priority: 2 Next Patient: John

1... To Add a Patient to Queue
2... To Get an Available  Patient
3... To Quit Program
1-3 :  3
```

รูปที่ 11

```
/*------ File : lldef.h ------*/
/* Header of linklist package */

/* structure of each element in list */
struct LINKTYPE
{
  struct LINKTYPE *next;
  struct LINKTYPE *previous;
  char *item;
};

/* collection of variable in linklist */
struct LINKLIST
{
  struct LINKTYPE *clp;    /* Current link Pointer   */
  struct LINKTYPE *head;   /* First link             */
  struct LINKTYPE *tail;   /* Tail link              */
  int listlength;        /* Number of links in list */
  int itemlength;
  int (*match) ();
};
/*-----------------------------------------------
 1.  Set linklist pakage to work with a new list.
*/
void LLsetlist (struct LINKLIST *new_list);

/*-----------------------------------------------
 2.  Set storage requirements of each item in list.
*/
void LLsetsize (int size);

/*-----------------------------------------------
 3.  Set matching function for searching an item.
*/
void LLmatch (int (*numatching) ());

/*-----------------------------------------------
 4.  Initialize the linked list with first item
*/
void LLinit (char *newitem);

/*-----------------------------------------------
 5.  Set current link pointer (clp) to head of list
*/
void LLhead ();

/*-----------------------------------------------
 6.  Set current link pointer (clp) to tail of list
*/
void LLtail ();

/*-----------------------------------------------
 7.  Move the clp down the list
*/
int LLnext ();

/*-----------------------------------------------
```

รูปที่ 11 (ต่อ)

```
 8.  Set the clp to the prevoius link
*/
int LLprevious ();

/*-----------------------------------------------
 9.  Retrieve the item for the clp link
*/
void LLretrieve (char *newitem);

/*-----------------------------------------------
 10. Add a new item in list following the clp,
     and reset clp to new link
*/
void LLadd (char *newitem);
```

```
/*-----------------------------------------------
 11. Add a new item at head, resset clp
*/
void LLaadhead (char *newitem);

/*-----------------------------------------------
 12. Delete and free the clp, reset clp to head
*/
void LLdelete ();

/*-----------------------------------------------
 12. Return the length (or number item) of the list
*/
int LLlength ();
```

รูปที่ 12

```
/*---- File : linklist.c -----*/
/* Module of linklist package */
#include <stdio.h>
#include <string.h>
#include <malloc.h>
#include "lldef.h"

/* Globol variable uses in linklist modlue */
static struct LINKLIST *list;

#define moveitem(dest,src)  memcpy(dest,src,list->itemlength);

/* Set this module to work with a new list */
void LLsetlist (new_list)
struct LINKLIST *new_list;
{
  list = new_list;
}

void LLsetsize (size)                /* Set storage requirements. */
int size;
{
  list->itemlength = size;
}

/* Set matching function */
void LLmatch (numatching)
int (*numatching) ();
{
  list->match = numatching;
}

/* Set clp to desired link.
  return TURE if found, FALSE otherwise */
int LLcheck (lookfor)
char *lookfor;
{
  for (;;)
    {
    if ((*list->match) (list->clp->item, lookfor))
      return (1);
    else
      if (!LLnext ())
        return (0);
    }
}

static struct LINKTYPE * LLcrlink ()  /* Allocate space for a link */
{
  struct LINKTYPE *link;
```

รูปที่ 12 (ต่อ)

```
  link = (struct LINKTYPE *)
      malloc (sizeof (struct LINKTYPE));
  link->item = malloc (list->itemlength);
  return (link);
}

/* Initialize the linked list with an item */
void LLinit (newitem)
char *newitem;
{
  struct LINKTYPE *LLcrlink();

  /* There is only one link */
  list->head = list->tail = list->clp = LLcrlink ();
  /* No next or previous */
  list->clp->next = list->clp->previous = NULL;
  /* Put this item into link */
  moveitem (list->clp->item,newitem);
  list->listlength = 1;
}

void LLhead ()
{
  list->clp = list->head;
}

void LLtail ()
{
  list->clp = list->tail;
}

int LLnext ()                    /* Move the clp down the list */
{
  if (list->clp->next == NULL)
    return (0);                 /* Return FALSE if at end of list */
  else
    {
    list->clp = list->clp->next;
    return (1);                 /* Return TURE if not at end */
    }
}

int LLprevious ()                  /* Set the clp to the prevoius link */
{
  if (list->clp->previous == NULL )
    return (0);                  /* Return FALSE if at start of list */
  else
    {
    list->clp = list->clp->previous;          /* Returen TURE if at the start of list */
    return (1);
    }
}

void LLretrieve (newitem)   /* Retrieve the item for the clp link */
char *newitem;
{
  moveitem (newitem ,list->clp->item);
}

void LLadd (newitem)
char *newitem;        /* Item to place in list */
{
  struct LINKTYPE  *newlink;

/* Creat new link
  ------------- */
  newlink = LLcrlink ();
  moveitem (newlink->item,newitem);
  list->listlength++;
```

รูปที่ 12 (ต่อ)

```
/* Reset pointer
   ------------ */
   newlink->next = list->clp->next;
   newlink->previous = list->clp;
   if (list->tail == list->clp)
     list->tail = newlink;
   else
     list->clp->next->previous = newlink;
   list->clp->next = newlink;
   list->clp = newlink;    /* Poin clp to new link */
}

/* Add a new link to the head of the list */
void LLaddhead (newitem)
char *newitem;  /* Item to add */
{
  struct LINKTYPE *newlink;

  if (LLlength () == 0)
    {
    LLinit (newitem);
    return;
    }
/* Creat new link
   -------------- */
  newlink = LLcrlink ();
  moveitem (newlink->item,newitem);
  list->listlength++;

/* Reset pointer
   ------------ */
  newlink->previous = NULL;
  newlink->next = list->head;
  list->head->previous = newlink;
  list->clp = list->head = newlink;
}

/*****************************************
 *  Delete and free the CLP, reset CLP to head
 *****************************************/
void LLdelete ()
{
  struct LINKTYPE *before,*after;

/* Is this the only link ?
   ---------------------- */
  if (list->head == list->clp && list->tail == list->clp)
    {
    list->head = list->tail = NULL;
    }

/* Is this the head ?
   ------------------ */
  else if (list->head == list->clp)
    {
    list->head = list->head->next;
    list->head->previous = NULL;
    }
/* Is this th tail
   --------------- */
  else if (list->tail == list->clp)
    {
    list->clp = list->tail = list->tail->previous;
    list->tail->next = NULL;
    }
/* Otherwise, it must be inside the list
   ------------------------------------ */
  else
    {
    before = list->clp->previous;
    after = list->clp->next;
```

รูปที่ 12 (ต่อ)

```
before->next = after;
after->previous = before;
}
/* Delete CLP
   ---------- */
   free (list->clp->item);
   free (list->clp);
   list->clp = list->head;
   list->listlength--;
}

int LLlength ()      /*  Return the length of the list */
{
   return (list->listlength);
}
```

```
/*---- File : queuedef.h ----*/
/* header file of Queue package */

#include "lldef.h"


#define Qset            LLsetlist /* Set current queue */
#define Qitemsize         LLsetsize /* define size of item in queue */
#define Qlength         LLlength    /* Return number of item in queue */

#define QUEUE           LINKLIST



/*----- Prototype of Queue pakage ----*/

void Qinsert (char *newitem);
void Qremove (char *newitem);
```

ที่ 13

```
/*---- File : queue.c ----*/
/* Module of queue package */
#include <stdio.h>
#include "queuedef.h"

/* Globol variable uses in queue modlue */
static struct QUEUE *q;

void Qinsert (newitem)
char *newitem;
{
   if (LLlength () == 0)
      LLinit (newitem);
   else
      LLaddhead (newitem);
}

void Qremove (olditem)
char *olditem;
{
   LLtail ();
   LLretrieve (olditem);
   LLdelete ();
}
```

รูปที่ 14

สรุป

ในฉบับนี้ได้เสนอตัวอย่างการนำแพคเกจลิงก์ลิสต์มาพัฒนาเป็นแพคเกจคิว ซึ่งแพคเกจคิวที่ได้จะมีคุณสมบัติเป็น Reusable ด้วยเช่นกัน ในตอนต่อไปของมุมภาษาซีในฉบับหน้าจะนำแพคเกจลิงก์ลิสต์ประยุกต์เข้ากับเรื่องอื่นอีกต่อไป และผู้เขียนคิดว่าผู้อ่านสามารถเก็บแพคเกจต่าง ๆ เหล่านั้นไปช่วยในการพัฒนาโปรแกรมของตนได้ในอนาคต

ก่อนจบผู้เขียนขอเรียนให้ทราบว่ามุมภาษาซีได้ปรับปรุงแพคเกจลิงก์ลิสต์จนสมบูรณ์แล้ว และต้องขออภัยต่อผู้อ่านเนื่องจากซอร์สโค้ดที่ลงพิมพ์ในตอนที่แล้วมีข้อผิดพลาดเล็กน้อย ดังนั้นผู้เขียนจึงขออนุญาตแก้ไขซอร์สโค้ดจากฉบับที่ 65 ซึ่งเป็นแพคเกจลิงก์ลิสต์ที่ถูกต้องดังรูปที่ 11 และรูปที่ 12 แต่สำหรับผู้อ่านที่ได้รับซอร์สโค้ดผ่านทางซอฟต์เมล์นั้นเป็นเซอร์สโค้ดที่ถูกต้องแล้ว ซึ่งผู้เขียนได้เพิ่มเติมตัวอย่างการใช้งานแพคเกจลิงก์ลิสต์เอาไว้ในแผ่นดิสค์ด้วย